算法设计与分析

动态规划：矩阵链乘问题

教材：算法导论(第三版)

矩阵链乘-问题定义

- 给定n个矩阵的序列$< A_1, A_2, .., A_n >$，需要计算其积$A_1, A_2, .., A_n$，并使得计算成本(所需要的标量乘法数)最小
- 矩阵乘法满足结合律，故不同的括号化方案产生同样积

例：$A_1 \sim A_4$积有五种不同的括号化方案

$(A_1(A_2\ (A_3A_4)))$　　$(A_1((A_2A_3)A_4))$

$((A_1A_2)(A_3A_4))$　　$((A_1A_2)A_3)A_4)$

$((A_1(A_2A_3))A_4)$

矩阵链乘-计算成本分析

- 不同的括号化方案产生不同的计算成本(标量乘法次数)

  两矩阵相乘$A_{pq} \cdot B_{qr}$的标量乘法次数为$p \times q \times r$

  例：设A_1, A_2, A_3的维数分别为$10 \times 100, 100 \times 5, 5 \times 50$

  $(A_1(A_2A_3))$：　　A_2A_3—$100 \times 5 \times 50 = 25000$（结果维数：$100 \times 50$）

  $A_1(A_2A_3)$—$10 \times 100 \times 50 = 50000$

  $Total: 50000 + 25000 = 75000$

  $((A_1A_2)A_3)$：　　A_1A_2—$10 \times 100 \times 5 = 5000$（结果维数：$10 \times 5$）

  $(A_1A_2)A_3$—$10 \times 5 \times 50 = 2500$

  $Total: 5000 + 2500 = 7500$

- 原问题等价于求最优(计算成本最小)的括号化方案

穷举法

- 枚举所有括号化方案并从中挑选一个最优括号化方案

$P(n)$表示n个矩阵$< A_1, A_2, .., A_n >$进行链乘时的所有括号化方案的数量。

将该序列从A_k和A_{k+1}间划分为两子序列，然后分别将其括号化，可以得到以下递归式:

$$\mathrm{P}(n) = \begin{cases} 1 & n = 1 \\ \sum_{k=1}^{n-1} \mathrm{P}(k)\mathrm{P}(n-k) & n > 1 \end{cases}$$

例如, $P(2) = P(1) = 1$， $P(3) = P(1)P(2) + P(2)P(1) = 2$

当$n \geq 2$, 我们有

$$P(n) = P(1)P(n-1) + \sum_{k=2}^{n-2} P(k)P(n-k) + P(n-1)P(1) \geq 2P(n-1)$$

因此，当$n \geq 2$, 我们有$P(n) \geq 2P(n-1) \geq \cdots \geq 2^{n-2}P(2) = 2^{n-2}$

指数级时间复杂度！

动态规划求解步骤

- Step1：定义子问题空间
- Step2：刻画最优解的结构特征（寻找最优子结构）
- Step3：根据最优子结构找到最优解的递归式
- Step4：根据递归式，自底向上地计算最优解的值
- Step5：根据计算最优解时得到的信息，构造一个最优解

动态规划的核心思想：

基于最优子结构特征(问题的最优解包含子问题的最优解)，自底向上计算最优解，并在计算过程中，通过记住所计算过的子问题的答案，避免重复计算相同子问题，从而降低时间复杂度

动态规划求解矩阵链乘问题：步骤1

- Step1：定义子问题空间
  - 对于任意的$i,j: 1 \le i \le j \le n$,将$A_i, A_{i+1}, .., A_j$的最优括号化方案定义为原问题的一个子问题，所有子问题构成子问题空间。
  - 子问题空间的大小尽可能小

    （显然，该子问题空间的大小至少为$\binom{n}{2} = O(n^2)$）
  - 子问题之间允许有重叠，例如$A_2, A_3, .., A_8$和$A_4, A_5, .., A_{11}$有重叠子问题$A_4, \ldots, A_8$

    （若无重叠子问题，则可用分治法求解。思考为什么）
  - 原问题是规模最大的一个子问题（$i = 1, j = n$）

动态规划求解矩阵链乘问题：步骤2

- Step2：刻画最优解的结构特征（寻找最优子结构）
  - 将$A_i, A_{i+1}, .., A_j$积简记为$A_{i..j}$，其中$(1 \le i \le j \le n)$，并假设A_t有p_{t-1}行p_t列（$1 \le t \le n$）
  - 当$i < j$，假设$A_i, A_{i+1}, .., A_j$的最优括号化方案在A_k和A_{k+1}之间进行分裂$(i \le k \le j-1)$，即最后一次矩阵相乘为$A_{i..k} \times A_{k+1..j}$（产生$z = p_{i-1} \times p_k \times p_j$次标量乘法）。
  - 则$A_{i..j}$的最优括号化方案必然是：用$A_i, A_{i+1}, .., A_k$的最优括号化方案计算$A_{i..k}$，用$A_{k+1}, A_{k+2}, .., A_j$的最优括号化方案计算$A_{k+1..j}$，然后将这两个积相乘产生积$A_{i..j}$

（反证法，见下页）

动态规划求解矩阵链乘问题：步骤2（续）

- Step2：刻画最优解的结构特征（寻找最优子结构）
  - 假设$A_i, A_{i+1}, .., A_j$的最优括号化方案在A_k和A_{k+1}之间进行分裂，则最优解为：

$$\underbrace{A_i, A_{i+1}, .., A_k}_{\text{最优括号化方案}} \times \underbrace{A_{k+1}, A_{i+1}, .., A_j}_{\text{最优括号化方案}}$$

证明：设$A_i, .., A_k$和$A_{k+1}, .., A_j$的最优括号化方案产生的标量乘法次数分别为m_1和m_2。注意$A_{i..k}$ 乘以$A_{k+1,..,j}$的标量乘法次数是固定的，为$z = p_{i-1} \times p_k \times p_j$。因此我们找到一个解，其乘法次数$= m_1 + m_2 + z$

但是，若最优解中$A_i, .., A_k$（或$A_{k+1}, .., A_j$）并非最优括号化方案，则意味着最优解的乘法次数$> m_1 + m_2 + z$, 得出矛盾。

动态规划求解矩阵链乘问题：步骤3

- Step3：根据最优子结构找到最优解的递归式
  - 设$m[i,j]$是计算$A_{i..j}$所需标量乘法的最小值(即子问题$A_{i..j}$的最小计算成本)，则$A_{1..n}$的最小计算成本是$m[1,n]$。
  - 初始条件：若$i=j$，$m[i,j]=0$（链上只有一个A_i，无需乘法）
  - 若$i<j$，由Step2中的最优解结构可知：
    假定最优括号化方案的分裂点为$k(i \le k \le j-1)$，则：

$$m[i,j]=m[i,k]+m[k+1,j]+p_{i-1}p_k p_j$$

由于不知道k的取值，可在$j-i$个分裂点中选取最优者。所以$A_{i..j}$的最小计算成本为：

$$m[i,j]=\begin{cases}0 & i=j\\ \min\limits_{i\le k<j}\{m[i,k]+m[k+1,j]+p_{i-1}p_k p_j\} & i<j\end{cases}$$

动态规划求解矩阵链乘问题：步骤3（续）

- Step3：根据最优子结构找到最优解的递归式（举例）

$$m[i,j]=\begin{cases}0 & i=j\\ \min\limits_{i\le k<j}\{m[i,k]+m[k+1,j]+p_{i-1}p_kp_j\} & i<j\end{cases}$$

设A_1,A_2,A_3的维数分别为$10\times100,100\times5,5\times50$。单个矩阵最小代价：

$$m[1,1]=m[2,2]=m[3,3]=0$$

两个矩阵最小代价：$m[1,2]=m[1,1]+m[2,2]+10\times100\times5=5000$

$$m[2,3]=m[2,2]+m[3,3]+100\times5\times50=25000$$

三个矩阵最小代价：

$$m[1,3]=\min\begin{cases}m[1,1]+m[2,3]+10\times100\times50=75000\\ m[1,2]+m[3,3]+10\times5\times50=7500\end{cases}=7500$$

可以看出，所计算的子问题规模从小变大，从而避免重复计算，降低时间复杂度

动态规划求解矩阵链乘问题：步骤4 (核心思想)

- Step4：根据递归式，自底向上地计算最优解的值
  - 回顾Step1中划分的子问题个数为$O(n^2)$。自底向上计算最优解的值，并将所计算过的子问题的解保存起来，可以保证每个子问题只被计算一遍。
  - 算法核心思想：从最小规模的子问题开始计算，根据Step3的递归式逐步计算规模更大的子问题，直到求出原问题的最优解。

具体步骤：

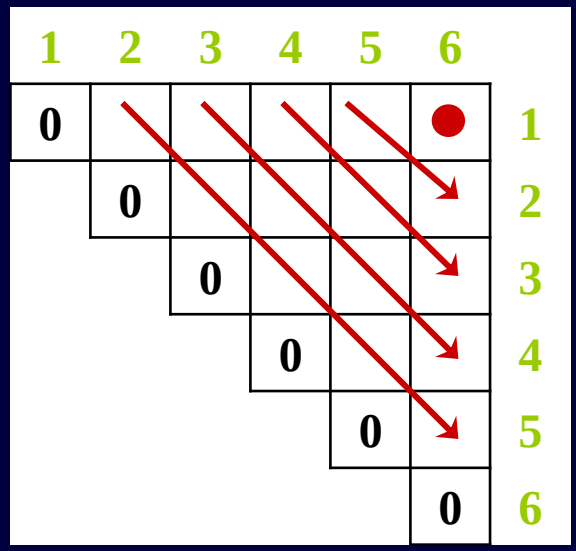
--首先设置$m[i,i]=0, i=i,..,n$

--随后计算$m[1,2], m[2,3],..,m[n-1,n]$

--然后是$m[1,3], m[2,4],..,m[n-2,n]$

--... 直到我们能够计算$m[1,n]$

$$m[i,j]=\begin{cases}0 & i=j\\ \min\limits_{i\le k<j}\{m[i,k]+m[k+1,j]+p_{i-1}p_kp_j\} & i<j\end{cases}$$



动态规划求解矩阵链乘问题：步骤4 (程序设计)

- Step4：根据递归式，自底向上地计算最优解的值

  - 算法输入：

    $p = (p_0, p_1, \dots p_n)$：其中$p_{i-1} \times p_i$是矩阵$A_i$的维数

  - 算法主要数据结构：

    二维矩阵$m[1 \dots n, 1 \dots n]$: 其中$m[i, j]$记录$A_{i \dots j}$的最优解所产生的标量乘法数（最优成本）；

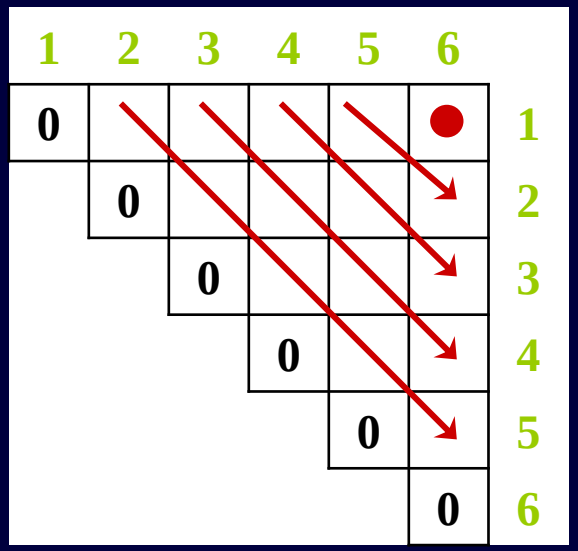
    二维矩阵$S[1 \dots n, 1 \dots n]$: 其中S$[i, j]$记录$A_{i \dots j}$的最佳分裂点（该最佳分裂点实现了计算$m[i, j]$的最优成本，后续用于构造最优解）

动态规划求解矩阵链乘问题：步骤4 (程序设计)

- Step4：根据递归式，

自底向上地计算最优解的值

$$m[i,j] = \begin{cases} 0 & i = j \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1}p_k p_j\} & i < j \end{cases}$$



--首先设置

$m[i,i] = 0, i = i,..,n$

--随后计算

$m[1,2], m[2,3], .., m[n-1,n]$

--然后是

$m[1,3], m[2,4], .., m[n-2,n]$

--… 直到我们能够计算

$m[1,n]$

```
MatrixChainOrder(p[]){
  n ← length[p] − 1;
  for i ← 1  to  n  do{
    m[i,i] ← 0;
  }
  for l ← 2  to  n  do{
    for i ← 1  to  n − l + 1  do{
      j ← i + l − 1;
      m[i,j] ← ∞;
      for k ← i  to  j − 1  do{
        q ← m[i,k] + m[k + 1,j] + p_{i−1} p_k p_j;
        if q < m[i,j]  then{
          m[i,j] ← q; /* 纪录最小代价 */
          S[i,j] ← k; /* 纪录最佳分裂点 */
        }//endif
      }//endfor k
    }//endfor i
  }//endfor l
  return  m&S; }
```

l: 当前计算的子问题的规模

动态规划求解矩阵链乘问题：步骤4 (程序设计)

- Step4：根据递归式，

自底向上地计算最优解的值

❖算法复杂度分析：

时间复杂度：由于嵌套了三层for循环，$T(n)=O(n^3)$，这是一个多项式级别的复杂度。

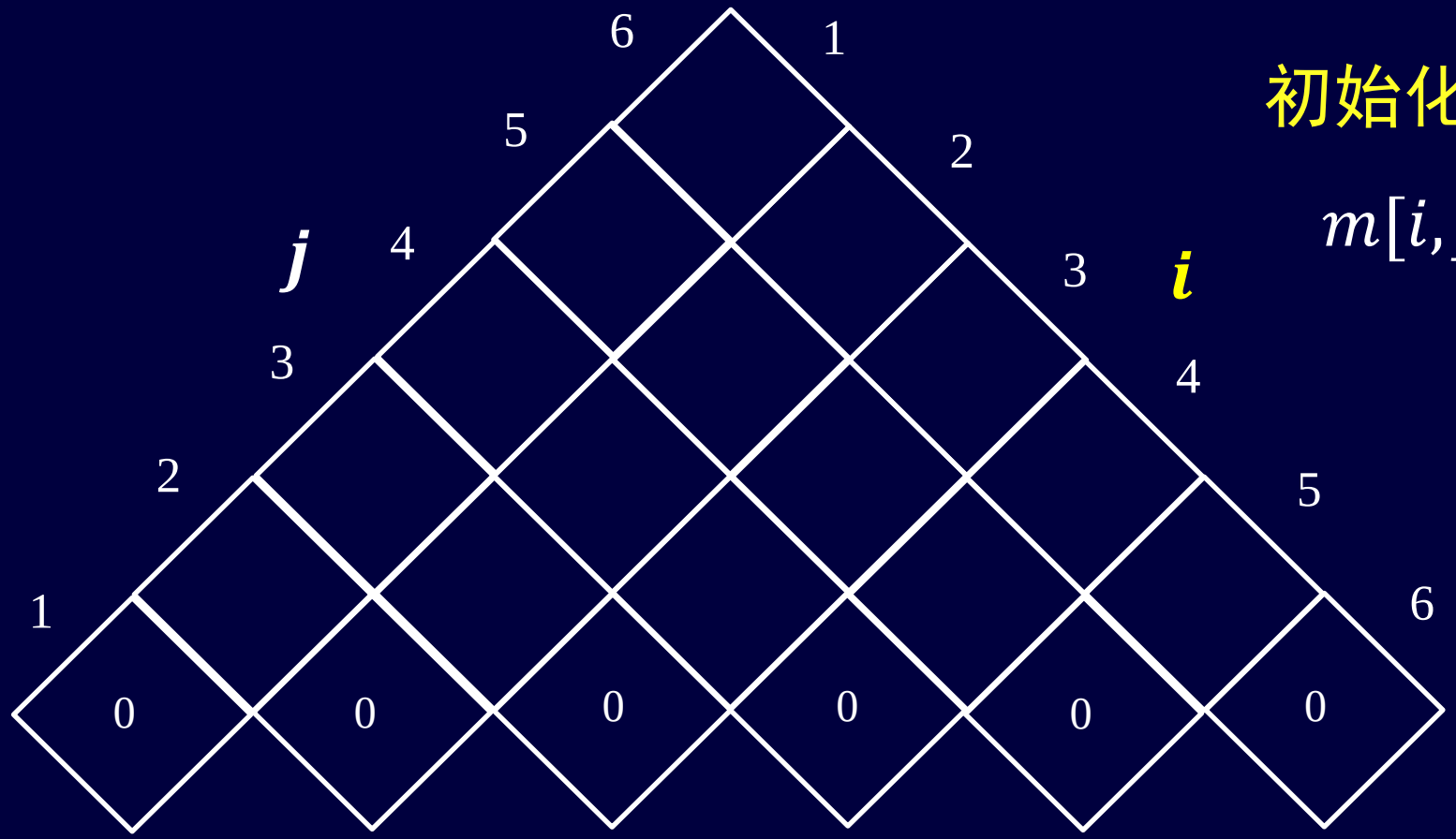
空间复杂度：因设置二维矩阵$m[1\ldots n,1\ldots n]$和二维矩阵$S[1\ldots n,1\ldots n]$，$S(n)=\Theta(n^2)$。

```
MatrixChainOrder(p[]){
  n ← length[p] − 1;
  for i ← 1  to  n  do{
    m[i,i] ← 0;
  }
  for l ← 2  to  n  do{
    for i ← 1  to  n − l + 1  do{
      j ← i + l − 1;
      m[i,j] ← ∞;
      for k ← i  to  j − 1  do{
        q ← m[i,k] + m[k + 1,j] + p_{i−1} p_k p_j;
        if q < m[i,j]  then{
          m[i,j] ← q; /* 纪录最小代价 */
          S[i,j] ← k; /* 纪录最佳分裂点 */
        }//endif
      }//endfor k
    }//endfor i
  }//endfor l
  return  m&S; }
```

l: 当前计算的子问题的规模

动态规划求解矩阵链乘问题：步骤4 (示例)

- Step4：根据递归式，自底向上地计算最优解的值



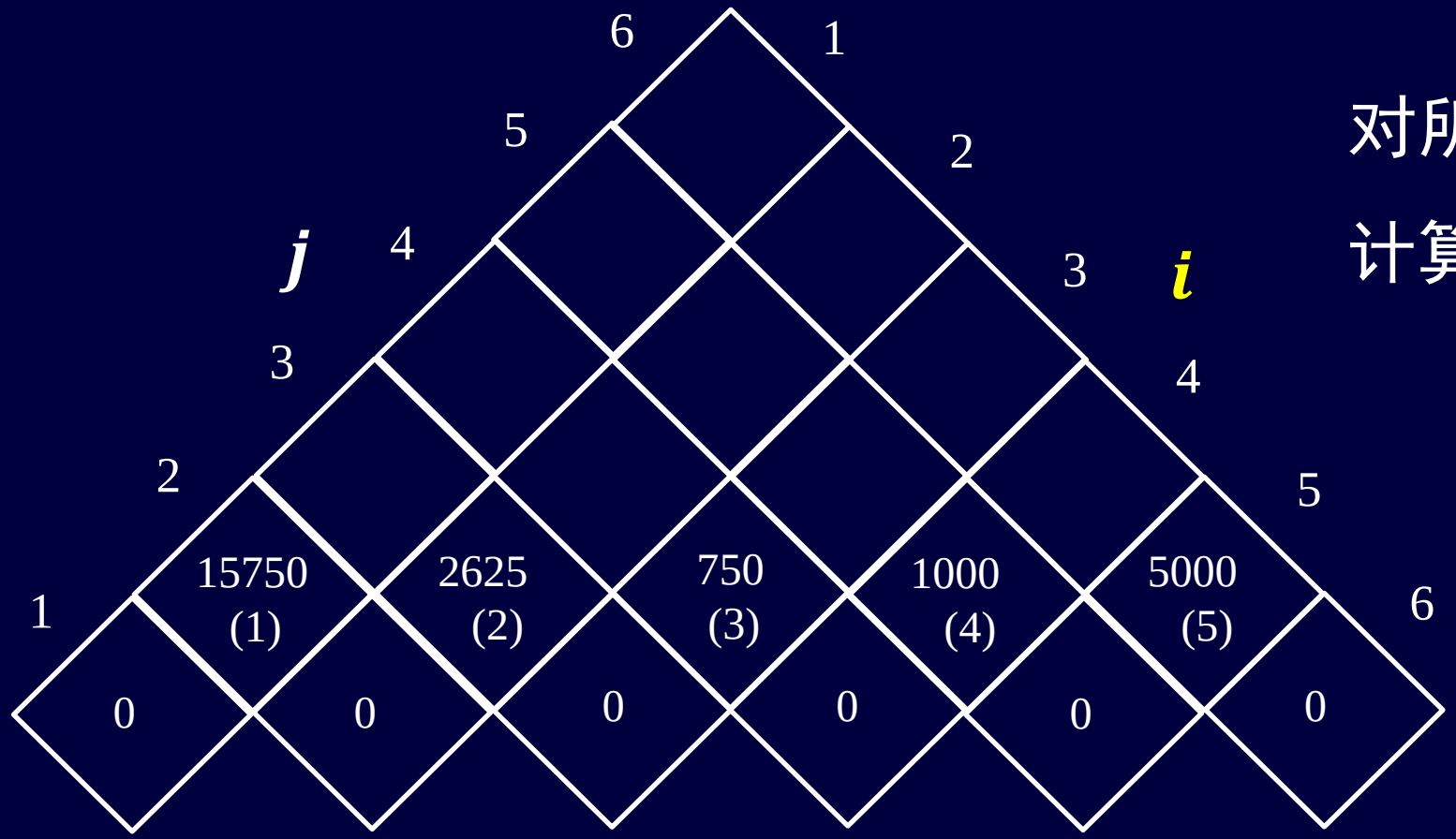
初始化：

$$m[i,j]=0;(1\le i=j\le n)$$

$A_1(30\times35)$ $A_2(35\times 15)$ $A_3(15\times5)$ $A_4(5\times10)$ $A_5(10\times20)$ $A_6(20\times 25)$

动态规划求解矩阵链乘问题：步骤4 (示例)

- Step4：根据递归式，自底向上地计算最优解的值



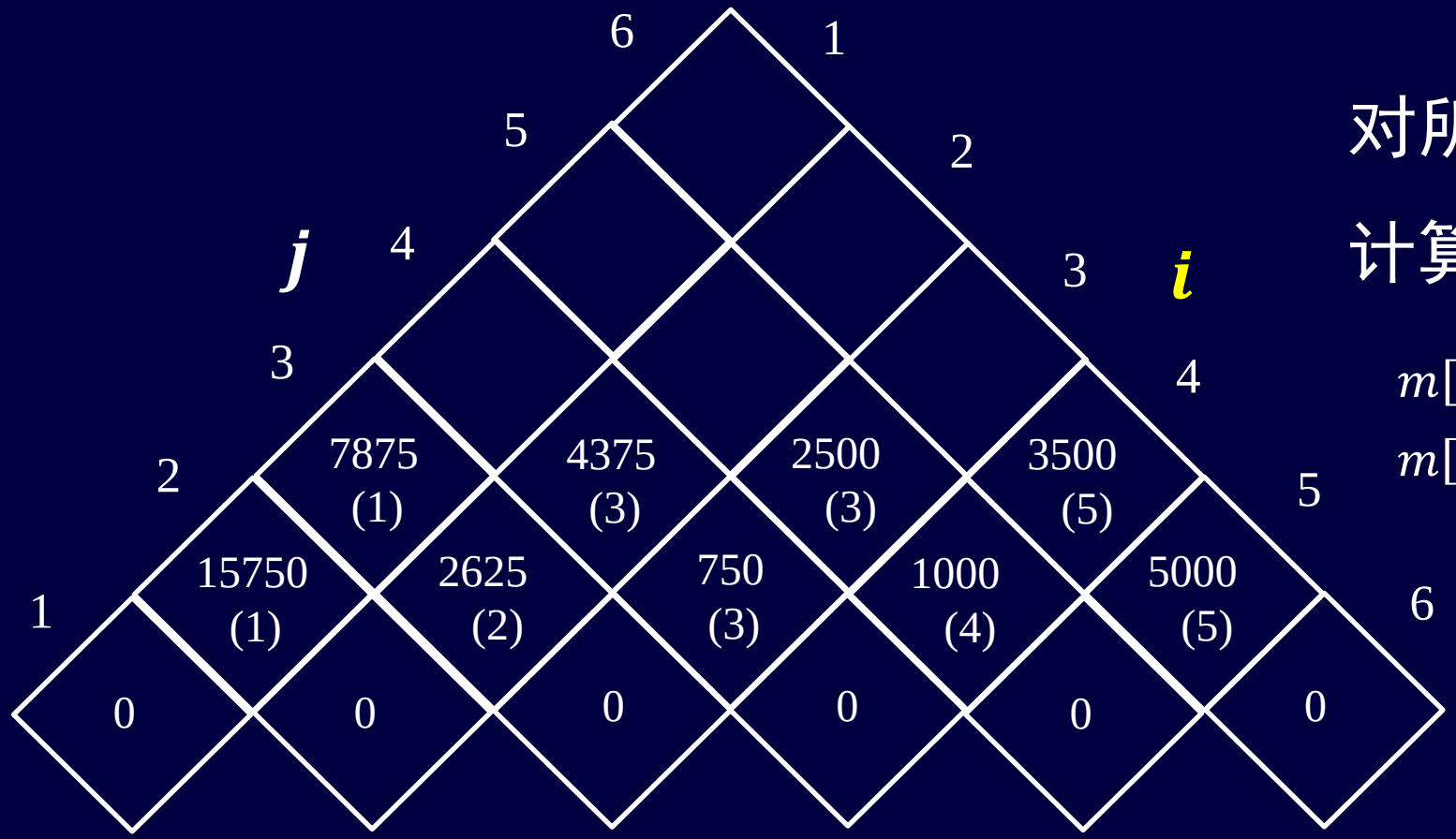
$A_1(30\times35)$ $A_2(35\times 15)$ $A_3(15\times5)$ $A_4(5\times10)$ $A_5(10\times20)$ $A_6(20\times 25)$

对所有满足$j = i + 1$的i, j:

计算$m[i, j]$; 令$S[i, j] = i$

动态规划求解矩阵链乘问题：步骤4 (示例)

- Step4：根据递归式，自底向上地计算最优解的值



$A_1(30\times35)$ $A_2(35\times 15)$ $A_3(15\times5)$ $A_4(5\times10)$ $A_5(10\times20)$ $A_6(20\times 25)$

对所有满足$j = i + 2$的i, j:

计算$m[i, j]$和$S[i, j]$。例：

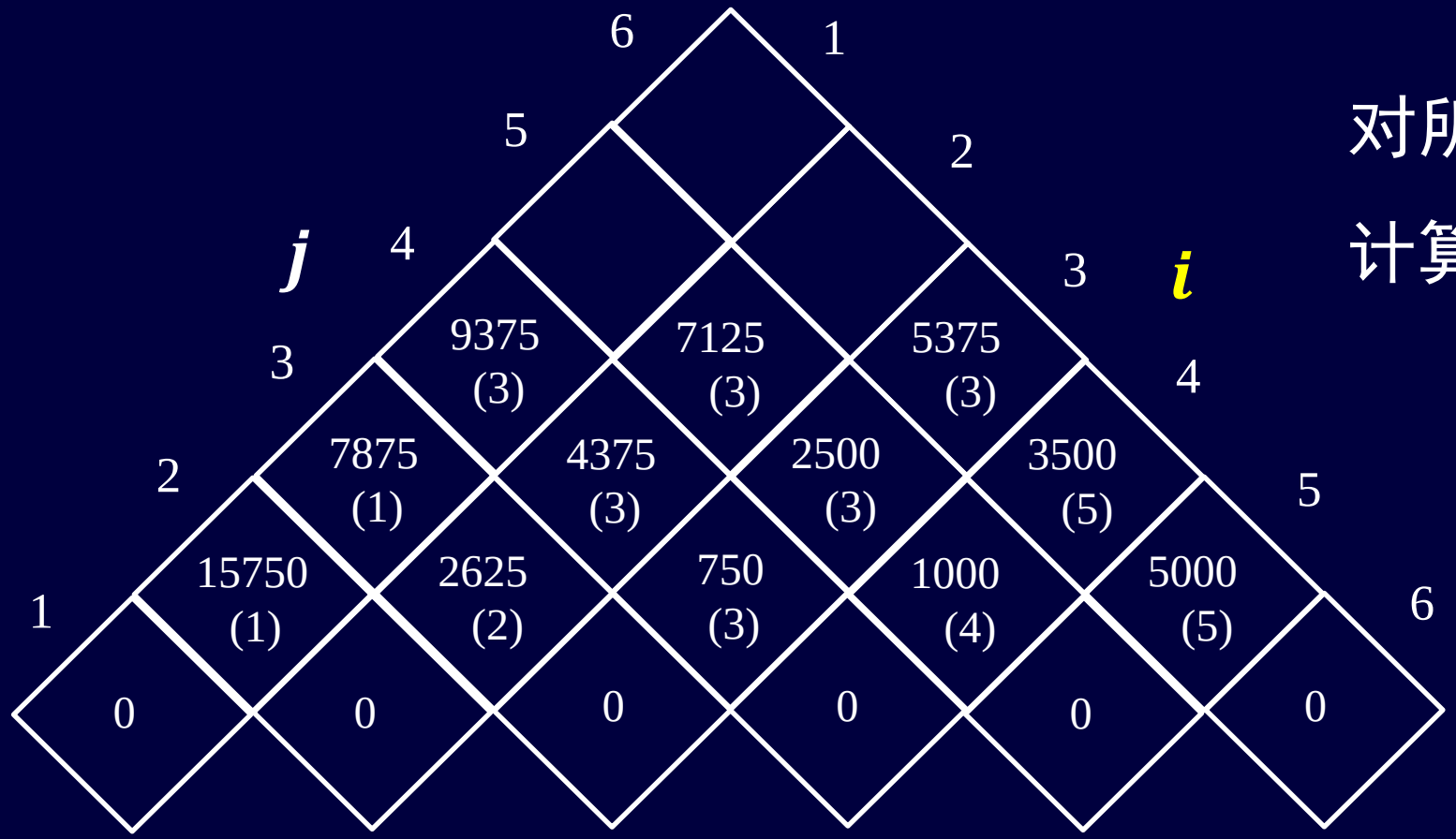
$$m[1,2] + m[3,3] + p_0p_2p_3 = 18000$$
$$m[1,1] + m[2,3] + p_0p_1p_3 = 7875$$

因此：

$$m[1,3] = 7875;\ S[1,3] = 1$$

动态规划求解矩阵链乘问题：步骤4 (示例)

- Step4：根据递归式，自底向上地计算最优解的值



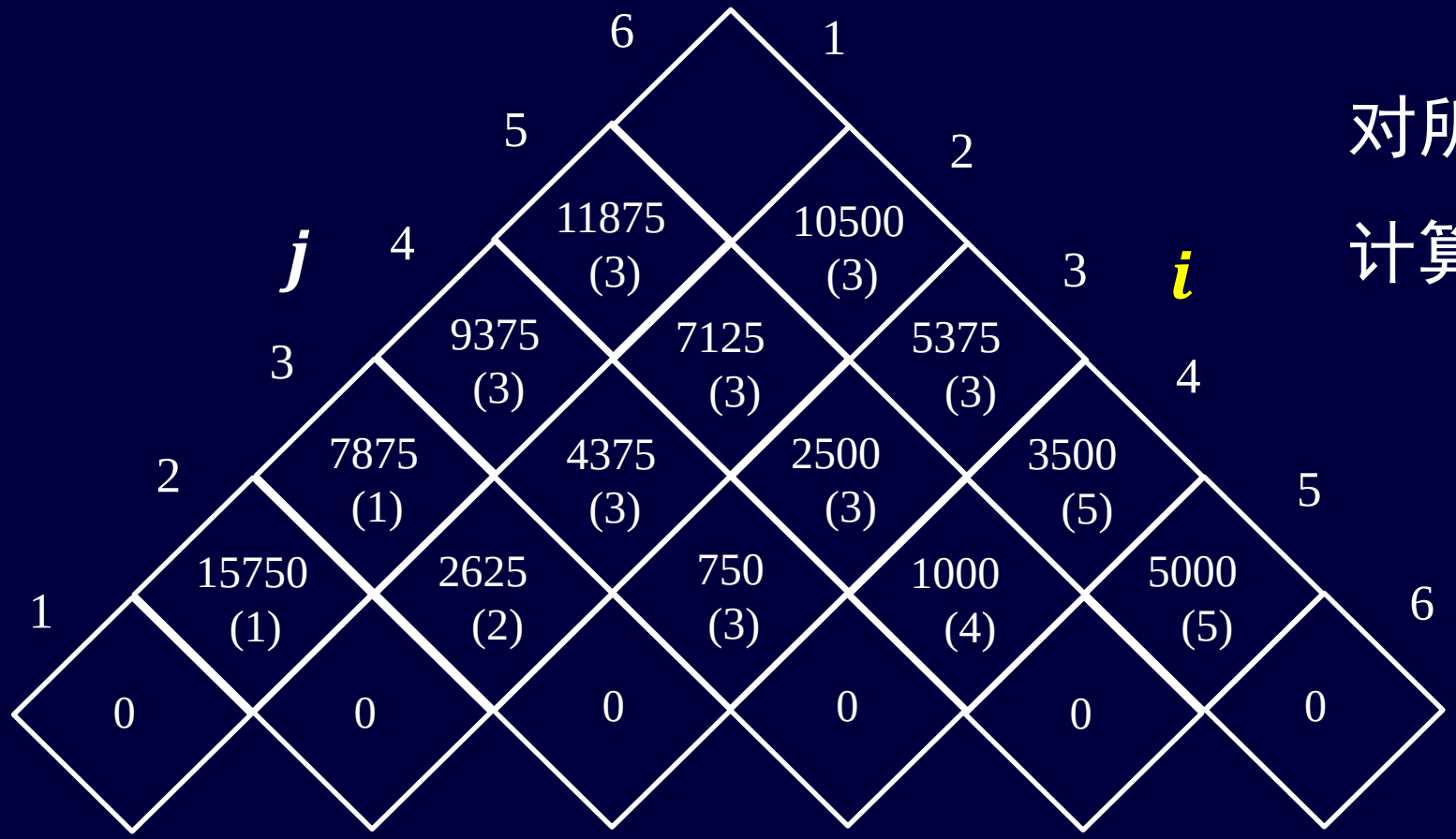
$A_1(30\times35)$ $A_2(35\times 15)$ $A_3(15\times5)$ $A_4(5\times10)$ $A_5(10\times20)$ $A_6(20\times 25)$

对所有满足$j = i + 3$的i, j:

计算$m[i, j]$和$S[i, j]$。

动态规划求解矩阵链乘问题：步骤4 (示例)

- Step4：根据递归式，自底向上地计算最优解的值



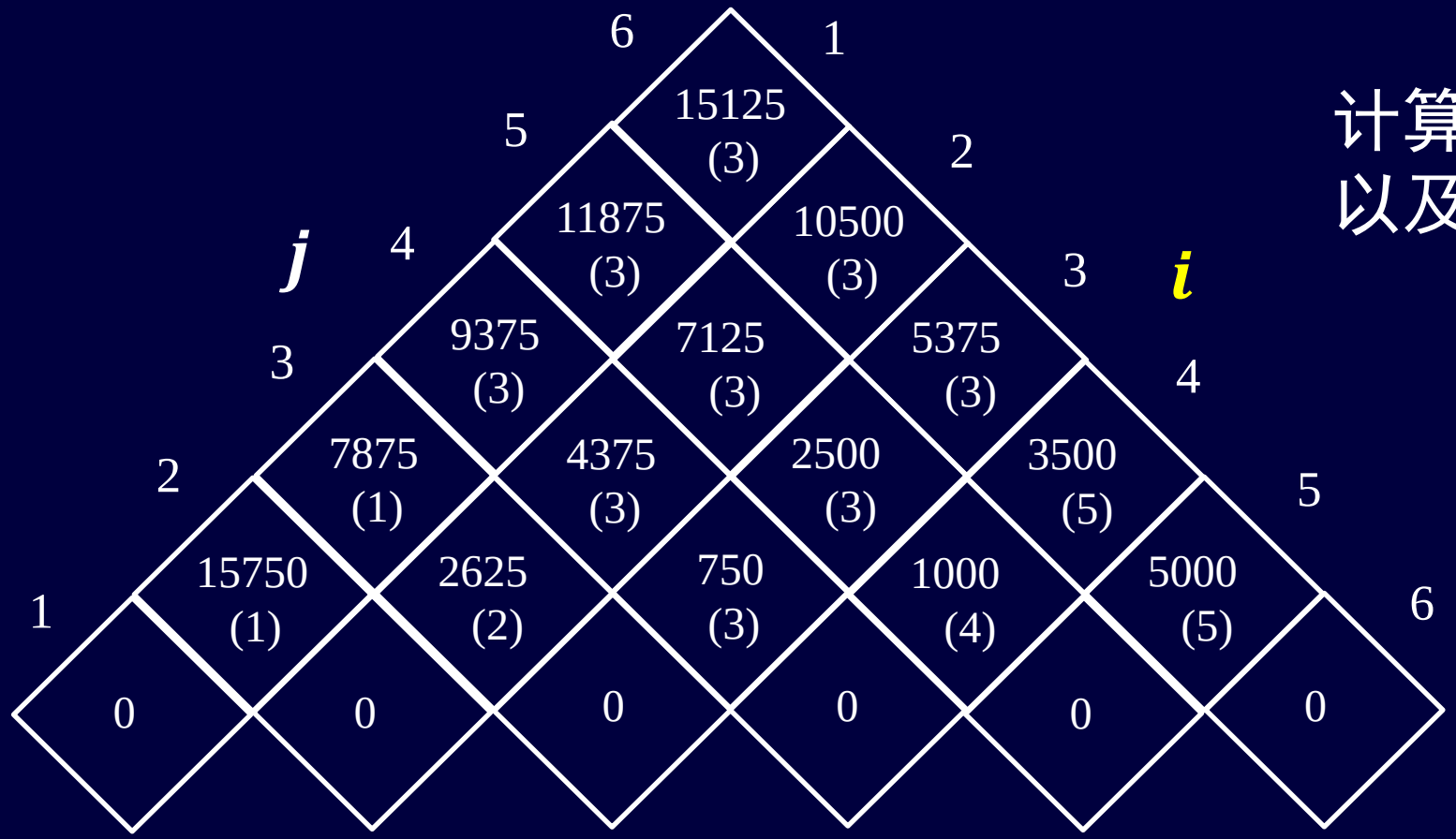
$A_1(30\times35)$ $A_2(35\times 15)$ $A_3(15\times5)$ $A_4(5\times10)$ $A_5(10\times20)$ $A_6(20\times 25)$

对所有满足$j = i + 4$的i, j:

计算$m[i, j]$和$S[i, j]$。

动态规划求解矩阵链乘问题：步骤4 (示例)

- Step4：根据递归式，自底向上地计算最优解的值



$A_1(30\times35)$ $A_2(35\times 15)$ $A_3(15\times5)$ $A_4(5\times10)$ $A_5(10\times20)$ $A_6(20\times 25)$

计算最优解$m[1,6]$
以及$S[1,6]$

动态规划求解矩阵链乘问题：步骤5

- Step5：根据计算最优解时得到的信息，构造一个最优解
  - 回顾$S[i,j]=k$记录了$A_{i..j}$最优括号化方案分裂点为k，因此

$$A_{i..j}=A_{i..k}\times A_{k+1..j}$$

```
PRINT − OPTIMAL − PARENS(s, i, j){
  if  i = j then
   print "A_i"
  else {
   print "("
   PRINT − OPTIMAL − PARENS(s, i, s[i, j])
   PRINT − OPTIMAL − PARENS(s, s[i, j] + 1, j)
   print ")"
  }}
```

打印最优括号化方案

调用Print-Optimal-Parens(s,1,n), 自顶向下递归, 打印最优括号化方案